Running Programs 11

Running a program as you work on it helps you find bugs and fix problems quickly. This chapter shows you how to run a program from the THINK Project Manager. It also tells you what happens when the THINK Project Manager can't run your program, how to compile files without running your program, how to examine what the compiler does with your code, and how to precompile headers so your projects compile more quickly.

What you should know

To run a program in the THINK Project Manager, you need a project document. If you don't know about projects, read Chapter 8, "Organizing Projects." If you want to experiment as you read this chapter, use one of the projects included in your package or one that you created from the tutorials.

Contents

Running a Program 199
    A word of caution 201

When Something Goes Wrong 201
    The bug alert 201
    The Compile Errors window 202
    The Link Errors window 205

Compiling Source Files 206
    Compiling files not in the project 206
    Compiling files already in the project 206
    Checking files without compiling 206
    Compiling files that auto-make misses. 206

Examining What the Compiler Does. 209
    Viewing the preprocessor output 209
    Disassembling your code 210
    Generating a link map 210

Precompiling Headers 212
Creating your own precompiled headers 213

Running a Program

The **Run** command in the **Project** menu lets you run your application project as you work on it. When you choose the **Run** command, the THINK Project Manager examines your project to see which files need to be recompiled and re-linked. If you've edited any source files, or if you've added new source files to the project, the THINK Project Manager marks them for compilation. If you changed a header file, the THINK Project Manager marks all the files that use it. If any files are marked, the THINK Project Manager asks you if you want to bring the project up to date.

Figure 11-1 Bring the project up to date dialog

If you click Update, the THINK Project Manager compiles the changed source files before running your program. If you click Don't Update, the THINK Project Manager runs your program with the object code that's in your project. If you click Make, the THINK Project Manager brings up the **Make** dialog, described later in this chapter. This lets you choose which files to compile. If you click Cancel, the THINK Project Manager doesn't run your program.

If you ask the THINK Project Manager to bring the project up to date, the THINK Project Manager compiles each file it marked and then links your program. It resolves references between files, and

makes sure that names are not missing or defined more than once. While it compiles and links, it displays a dialog showing its progress.

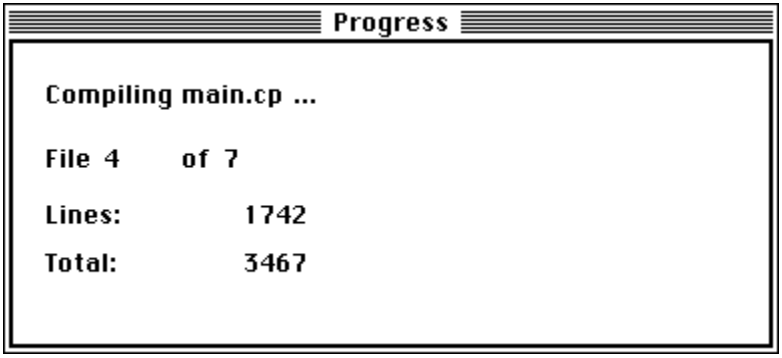

Figure 11-2 The Progress dialog

The dialog shows you the file it is compiling, the number of files it has compiled so far, the number of lines it has compiled in this file, and the total number of lines it has compiled in other files. If you're running under System 7, you can use other programs while the THINK Project Manager is compiling in the background. To cancel compiling, press Command-. (Command-Period).

You can use the Confirm Saves option to tell the THINK Project Manager to always save your files without asking. It is described on "Saving files automatically" in Chapter 10.

After the THINK Project Manager links your program, it asks you if you want to save your files before it runs your program. If you click Don't Save and your program crashes, you could lose all the changes you made.

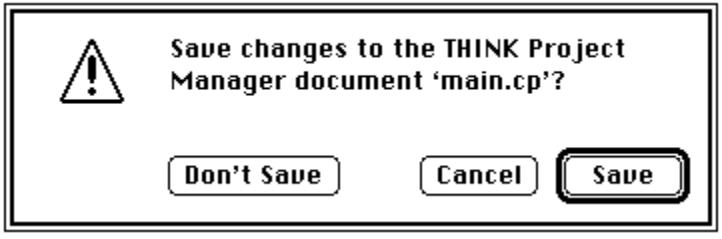

Figure 11-3 Asking whether to save your files

The THINK Project Manager now launches your application as if you had opened it from the Finder. If you're using System 7 or MultiFinder under System 6, your application runs in its own partition. You can switch back and forth between the THINK Project

Manager and your application. The **Run** command in the THINK Project Manager changes to **Resume**. Choosing **Resume** brings your application to the foreground. When you quit your application, you're back in the THINK Project Manager.

A word of caution

*You can make a backup of a project in the Finder. Use the **Duplicate** command in the **File** menu.*

When you're running under System 7 (or MultiFinder under System 6), you must be more careful about stray pointers and much more conscientious about backups.

Warning

You could destroy your project and all of your work if you don't program carefully and keep backups.

A pointer to random memory may be pointing into another application's partition. A stray pointer to the THINK Project Manager's data structures may damage your project, and the damage is copied out to disk. Be careful with stray pointers. If your project becomes corrupted, delete the damaged project, start over with your backup, and bring it up to date with the Use Disk button in the **Make** dialog.

When Something Goes Wrong

If something goes wrong while the THINK Project Manager is carrying out a command, it displays the reason in one of these places: a bug alert, the Compile Errors window, or the Link Errors window.

The bug alert

When you choose a command that the THINK Project Manger can't perform, it displays the reason in a bug alert like the one in Figure 11-4. To get rid of the alert, click the mouse or press the Enter or Return key.

Figure 11-4 A bug alert

When a THINK compiler comes across code it can't compile, it may display the reason in a bug alert. To get rid of the alert, click the

mouse or press the Enter or Return key. The THINK editor opens the source file and highlights the line that contains the error.

Note

Usually when a THINK compiler can't compile some code, it prints the reason to the Compile Errors window. It displays the reason in a bug alert only if you chose one of the following commands—**Check Syntax**, **Compile**, **Precompile**, **Disassemble**, or **Preprocess**—and the compiler is set up to stop compiling a file after the first error.

The Compile Errors window

When a THINK compiler comes across code it can't compile or believes is incorrect, it usually prints the reason to the Compile Errors window, as in Figure 11-5, and continues compiling your project.

Figure 11-5 The Compile Errors window

For a list of the messages that THINK C can produce, see "Error Messages," Appendix C, in this guide. For a list of error messages that Symantec C++ can produce, see Appendix C, "Error Messages" of the Symantec C++ Compiler Guide.

A message about code that the compiler can't compile is called an error message, and is marked `Error`. The compiler cannot compile the code until you change it. Common error messages are that your code is missing punctuation or that you're using a variable without declaring it.

A message about code that the compiler believes is incorrect is called a warning message, and is marked `Warning`. The compiler can compile code that generates a warning, but the warning probably points out code that doesn't do what you expect. Examples of common warning messages are that you aren't using a variable you declared or that you're using obsolete features of the language.

Note

THINK C does not print warning messages. It prints only error messages.

To see the line described in the error message, double-click the message, or select it and press Return. If the error is in a source file, the THINK editor opens the file and selects the line, as in Figure 11-6. The THINK Editor keeps track of your edits, and so it selects the correct line in the file after you've edited the file.

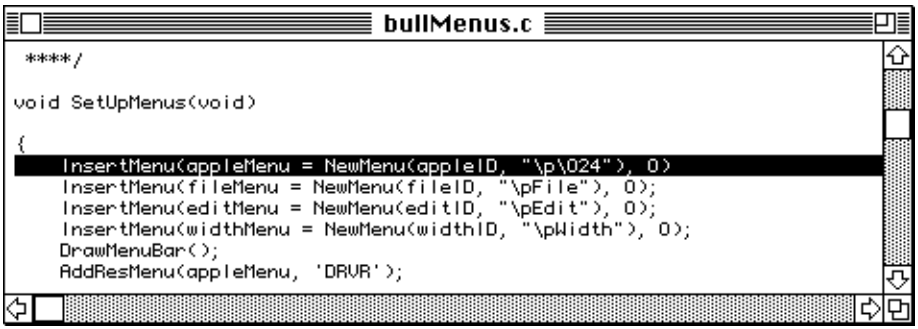

Figure 11-6 An error in a source file

If the error is in a compiler's project prefix, the THINK Project Manager opens the **Compiler Option** dialog to the Prefix page and selects the line, as in Figure 11-7.

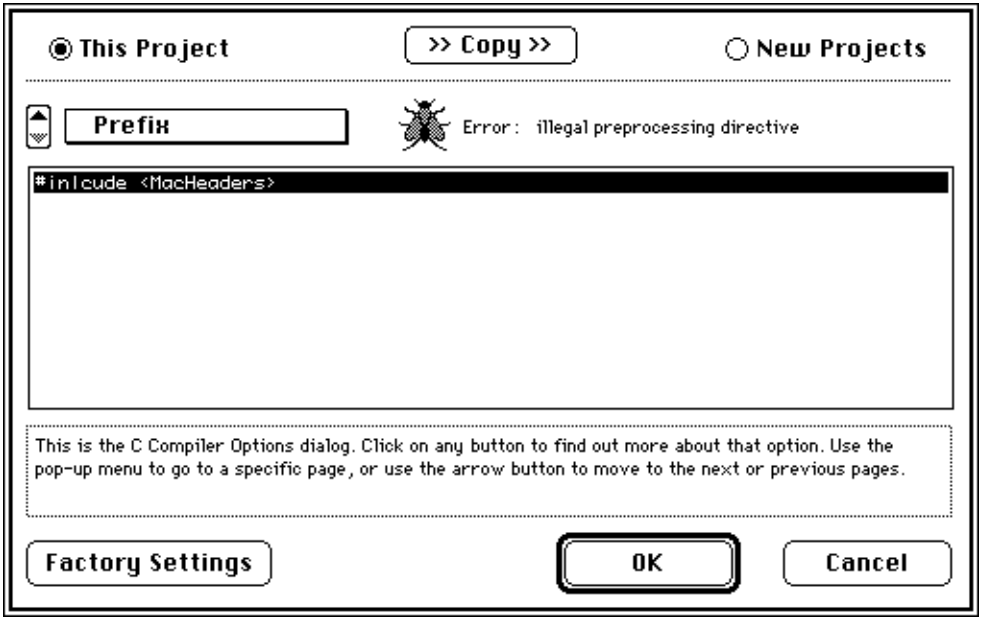

Figure 11-7 An error in the project prefix

To see the line for the next error in the Compile Errors window, choose **Go To Next Error** from the **Search** menu. To see the line for the previous error, choose **Go To Previous Error** from the **Search** menu.

When the Compile Errors window is active, you can scroll through the messages from the keyboard. To go to the next message, press Down Arrow or Tab. To go to the previous message, press Up Arrow or Shift-Tab. When you're at the last message, press Tab for the first message; when you're at the first message, press Shift-Tab for the last message.

You can copy and print the error messages. To copy a message to the Clipboard, select it and choose **Copy**. To copy all the messages to the clipboard, choose **Select All** and then **Copy**. To print the messages, choose **Print**.

After a compiler prints a warning or error message, it continues compiling. Symantec C++ lets you choose whether it continues to compile the current file or compiles the next file. By default, Symantec C++ continues compiling the current file until it prints a certain number of error messages. THINK C and THINK Rez do *not* continue compiling the current file, and move onto the next file automatically.

Note

If you close your project while the Compile Errors window is still open, the THINK Project Manager saves the window's contents and restores them when it opens the project again.

The Link Errors window

If the THINK Project Manager can't link the files in your project, it displays a bug alert saying `Link Failed` and shows you the Link Errors window. The messages in the Link Errors window tell you the names of undefined routines or the names of routines you've defined more than once.

Figure 11-8 Link Errors window

If you want to know in which files the THINK Project Manager found the errors, use the **Check Link** command in the **Project** menu. When you use this command, the THINK Project Manager reports the filenames as well as the link errors.

Note

If you close your project while the Link Errors window is still open, the THINK Project Manager saves the window's contents and restores them when it opens the project again.

Compiling Source Files

Unlike traditional compilers, the THINK compilers do not generate separate object files from your source files. Instead, the THINK compilers put all the object code into the project document. Although you can compile files manually, most of the time you'll use the auto-make facility to compile your files.

Compiling files not in the project

To learn how to add to or change these suffixes, see "Changing File Extensions" in Chapter 8.

You can add a source file to your project and compile it in one step. First, create your source file with the THINK Editor. Save your file in the same folder as the project document. Make sure that the file name ends in a suffix for one of your compilers. Table 11-1 shows the default suffixes.

Compiler	Suffix
THINK C	`.c` or `.asm`
Symantec C++	`.cp` or `.cpp`
THINK Rez	`.r`

Table 11-1 Suffixes for THINK Project Manager source files

Next, choose **Compile** from the **Source** menu. A dialog box shows you how many lines the THINK compiler has compiled. If no errors were in the source file, the THINK compiler adds the file and its object code to the project.

Compiling files already in the project

If you want to compile a file that is already in the project, just click its name in the Project window and choose **Compile** from the **Source** menu. Once a file is in the project, you don't need to open it to compile it.

Checking files without compiling

You can check the syntax of THINK C, Symantec C++, and THINK Rez files.

Sometimes you just want to make sure that your source file will compile without actually compiling it. The **Check Syntax** command in the **Source** menu checks the syntax of the contents of the frontmost edit window without generating code or adding the file to the Project window.

Compiling files that auto-make misses

The THINK Project Manager's auto-make facility can usually figure out which files need to be recompiled. It relies on the THINK Project

Manager's knowledge of when you last modified the files in the THINK Project Manager. However, it can guess wrong in these cases:

- If you modified a file outside THINK Project Manager; for example, if you edit a file in another editor that is not THINK-compatible.
- If a file that's shared by two projects is changed in only one. The second project won't realize the file has changed.
- If you moved a file to a different folder. For example, if you copy a project from your friend's hard disk to yours, auto-make won't find libraries that you keep in different folders.

If you think that the auto-make facility may be missing some files, you have two choices. You can select the Always Check File Dates option on the Preferences page of the **THINK Project Manager options** dialog or use the **Make** command.

If you choose the Always Check File Dates option, the THINK Project Manager checks the modification dates for your project's files when it runs or builds your project. It recompiles the files if their dates are later than the ones stored in the project for the files. This option is useful for the first two problems described above.

Make can search the THINK and project trees for a file, in addition to checking modification dates. It's useful for avoiding the problems

described above. When you choose **Make** from the **Project** menu, it displays the dialog in Figure 11-9.

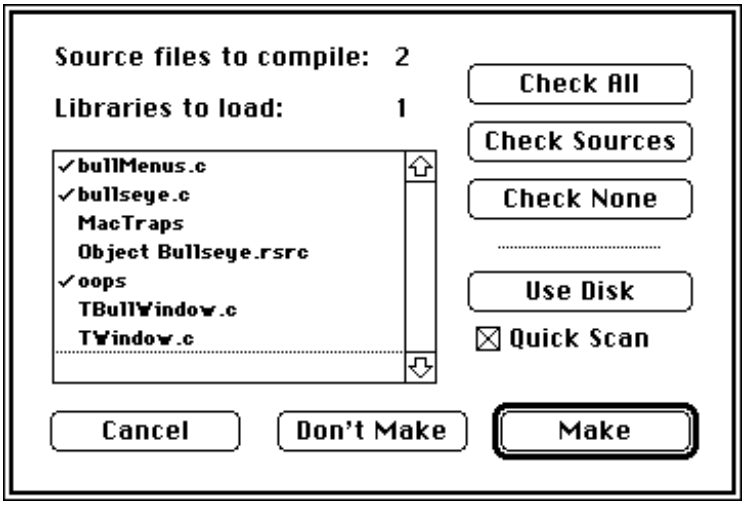

Figure 11-9 Make dialog

The scroll box shows all the files in the project. It puts a checkmark next to the files that the THINK Project Manager thinks need to be recompiled (or in the case of libraries, reloaded).

If **Make** misses some files you know need to be recompiled, use the cursor to check files in the scroll box, or click the Check All, Check Sources, or Check None button.

The Use Disk button examines all the files in your project and can check them in the scroll box for you. If the Quick Scan option is selected, Use Disk just examines the modification dates of your project's files. It checks files whose modification dates are later than the dates stored in the project. If the Quick Scan option is not selected, Use Disk also looks for files that aren't in the same directory they used to be in. It only checks the directory that the files were originally in. (The Use Disk feature can't find files when you move them from one tree to another. See "Moving Files on Disk" in Chapter 9 for more information.)

When you click Use Disk, it starts examining your files. If Quick Scan is not selected, Use Disk displays its progress in a dialog. You can abort it by typing Command-. (Command-Period), although the

next time you click Use Disk, it will start examining your files from the beginning.

When Use Disk is done, it updates the status of the files in the scroll box. If Use Disk can't find a source file or library, it's not checked. However, if a source file is found but one of the files it includes is not found, the source file is checked.

When you press the Make button, the THINK Project Manager recompiles and reloads the checked files and libraries. If you press the Don't Make button, the project is updated next time you use any of the commands that update the project: **Bring Up To Date**, **Run**, **Check Link**, or **Make**. If you click the Cancel button, your changes are not remembered.

Examining What the Compiler Does

The THINK Project Manager lets you look at your source code and finished applications in three different ways. You can see the preprocessor output of a source file, the assembly code a source file produces, and a link map of a finished application.

Viewing the preprocessor output

You can view the preprocessor output of THINK C, Symantec C++, and THINK Rez files.

If you think you have a bug in one of your macros, use the **Preprocess** command in the **Source** menu. It runs the code in the frontmost window through the preprocessor for the file's compiler and displays the result in a new window. The preprocessor expands your macros, includes the contents of your header files, and evaluates your conditional compilation statements (like `#ifdef` in C or C++). You can save and print the contents of this window like you would any other file.

Note

If you have the same code in source files for different translators, the **Preprocess** command may produce different results for each file. The **Preprocess** command asks the source code's translator to preprocess the code, and different translators may preprocess the same code differently.

You can disassemble THINK C and Symantec C++ files.

Disassembling your code

Looking at the assembly code that the compiler produces helps you debug your code and figure out how efficient it is. The **Disassemble** command in the **Source** menu disassembles the code in the frontmost window and displays the result in a new window. You can save and print the contents of this window like you would any other file.

Note

If you have the same code in source files for different translators, the **Disassemble** command may produce different results for each file. The **Disassemble** command asks the source code's translator to disassemble the code, and different translators may disassemble the same code differently.

Generating a link map

The THINK Project Manager can write a link map for your application. The link map lists all your project's segments, including one for global data. For each function (or global variable) in a segment, the map lists its name, its position in the segment, and the file it's defined in. To generate a link map, turn on the Generate Link Map option in the Preferences page of the **THINK Project Manager Options** dialog.

The THINK Project Manager creates the link map only when you use the **Build** command. The name of the map is the name of the project with `.map` appended. For example, the link map for the `Bullseye.π` project is `Bullseye.π.map`. The THINK Project Manager places the link map in the project folder and erases any link map that was there.

This is an excerpt from the global data section of a link map:

```
Segment "%GlobalData" size=$00084E
   appleMenu     -$00084E(A5) file="bullMenus.c"
   fileMenu      -$00084A(A5)
   editMenu      -$000846(A5)
   widthMenu     -$000842(A5)
   windowBounds  -$0007A8(A5) file="bullWindow.c"
   circleStart   -$0007A0(A5)
```

Here is how to read it:

- The top line gives the segment's name and size. This segment is named `%GlobalData` and is `0x00084E` bytes large.
- Each of the other lines lists a variable and its address as a negative offset from `A5`. The variable `appleMenu` is at `0x00084E` offset from `A5`.
- At the far right is the name of the file that defines the variable. The variables `appleMenu`, `fileMenu`, `editMenu`, and `widthMenu` are in the file `bullMenus.c`. The variables `windowBounds` and `circleStart` are in `bullWindow.c`.

If you're doing object-oriented programming in either THINK C or Symantec C++, you'll see some other entries in the global data section. In THINK C, you might see entries like these:

```
?TShape::Draw -$000078(A5)
TShape::      -$00005A(A5)
```

`?TShape::Draw` is a method dispatch table, which contains information about the method. The class name is the name of the first class that defined the method. `TShape::` is a class info record, which contains some information about the class.

In Symantec C++, you might see entries like these:

```
$TPasOb::Draw -$00029A(A5)
_TPasOb       -$000292(A5)
TBlue::_vtbl  -$00028E(A5)
```

`$TPasOb::Draw` and `_TPasOb` are a method dispatch table and a class info record. Symantec C++ generates these only for classes that use the Pascal Object model and their methods. `TBlue::_vtbl` is a v-table, which contains some information about the class. Symantec C++ generates these for virtual member functions.

This is an excerpt from a code segment:

```
Segment "Seg2" size=$0004FC rsrcid=2
   SetUpMenus        $000004   file="bullMenus.c"
   AdjustMenus       $000092
   HandleMenu        $000186
   InitMacintosh     $000282   file="bullseye.c"
   HandleMouseDown   $0002A6
   HandleEvent       $00036E
   main              $000414  JT=$000072(A5)
```

Here is how to read it:

- The top line gives the segment's name, size, and resource ID. The segment `Seg2` is `0x0004FC` bytes large and is in code resource 2.
- Each of the other lines lists a function name and its offset within the segment. The function `SetUpMenus()` is at offset `0x000004`.
- If a function has a jump table entry, the address of the entry is listed as an offset from `A5`. The function `main()` has a jump table entry at `0x000072` offset from `A5`.
- At the far right is the name of the file that defines the function. The functions `SetUpMenus()`, `AdjustMenus()`, and `HandleMenu()` are in the file `bullMenus.c`. The functions `InitMacintosh()`, `HandleMouseDown()`, `HandleEvent()`, and `main()` are in `bullseye.c`.

Precompiling Headers

You can precompile THINK C and Symantec C++ files.

The THINK Project Manager lets you precompile header files. (In C and C++, they're also called include files.) Precompiled headers contain only declarations and preprocessor symbols. Since precompiled headers are in a format the THINK Project Manager can use readily, they load faster than text header files.

With THINK C you can only precompile source files that end in `.c`. With Symantec C++ you can only precompile source files that end in `.c`, `.cp`, and `.cpp`.

Note

If you're using the source-level debugger, you should use precompiled headers. Precompiled headers help make the debugger tables much smaller.

Many THINK compilers include a precompiled header that contains the most common declarations used for writing Macintosh programs. THINK C includes `MacHeaders`, which is a precompiled version of `Mac #includes.c`, and Symantec C++ includes `MacHeaders++`, which is a precompiled version of `Mac #includes.cp`. If you include a precompiled header in the compiler's project prefix, the compiler automatically includes it in all the compiler's source files in your project, so you never have to explicitly include common header files like `QuickDraw.h`. (It doesn't hurt if you do.) By default, the project prefix for THINK C is `#include <MacHeaders>`, and for Symantec C++ is `#include <MacHeaders++>`.

Note

The THINK C precompiled header `MacHeaders` is compiled with the Strict Prototype Enforcement option turned on. To create a version with simplified prototypes, go to `Mac #includes.c`, and define the macro `SIMPLIFY_PROTOTYPES` to `1`.

You can't use one compiler's precompiled header in another compiler's source file. For example, a THINK C source file can't use `MacHeaders++`. If your project contains source files from more than one compiler and you want all files to use the same precompiled header, you'll have to precompile the header separately with each compiler, and include the appropriate version in each compiler's project prefix.

Creating your own precompiled headers

If you want to use your own precompiled headers, follow these steps:

1. Check the compiler's project prefix for any statements you want in your precompiled header. The project prefix isn't included in your precompiled header.

2. Create a file containing your declarations. Precompiled headers for THINK C and Symantec C++ usually contain a series of `#include` statements. Your header can include another precompiled header, such as `MacHeaders` or `MacHeaders++`, as long as it's in the first line of your precompiled header.

3. If you need to change the compile options, do it from inside the precompiled header's source file. In a THINK C source file, use the `#pragma options` directive. Otherwise, the compiler uses the settings in the compiler's options dialog and uses different options depending on the project that's open. For more information, see the *Symantec C++ Compiler Guide*.

4. Choose the **Precompile** command from the **Source** menu. When the THINK Project Manager is through precompiling, it asks you to name the file.

You use a precompiled header the same way you use any other header file. In THINK C or Symantec C++ source files, use the `include` statement to load it into your source file. The `#include` statement must be the first non-comment line of your source file. You can use only one precompiled header per source file. (If you #include a precompiled header in the compiler's project prefix, you can't include any other precompiled header.)

If you don't include any precompiled headers in a compiler's project prefix, you can use several different precompiled headers for different parts of your program, and you can still explicitly include `MacHeaders` or `MacHeaders++` for certain files.

Note

You can use only one precompiled header per source file.